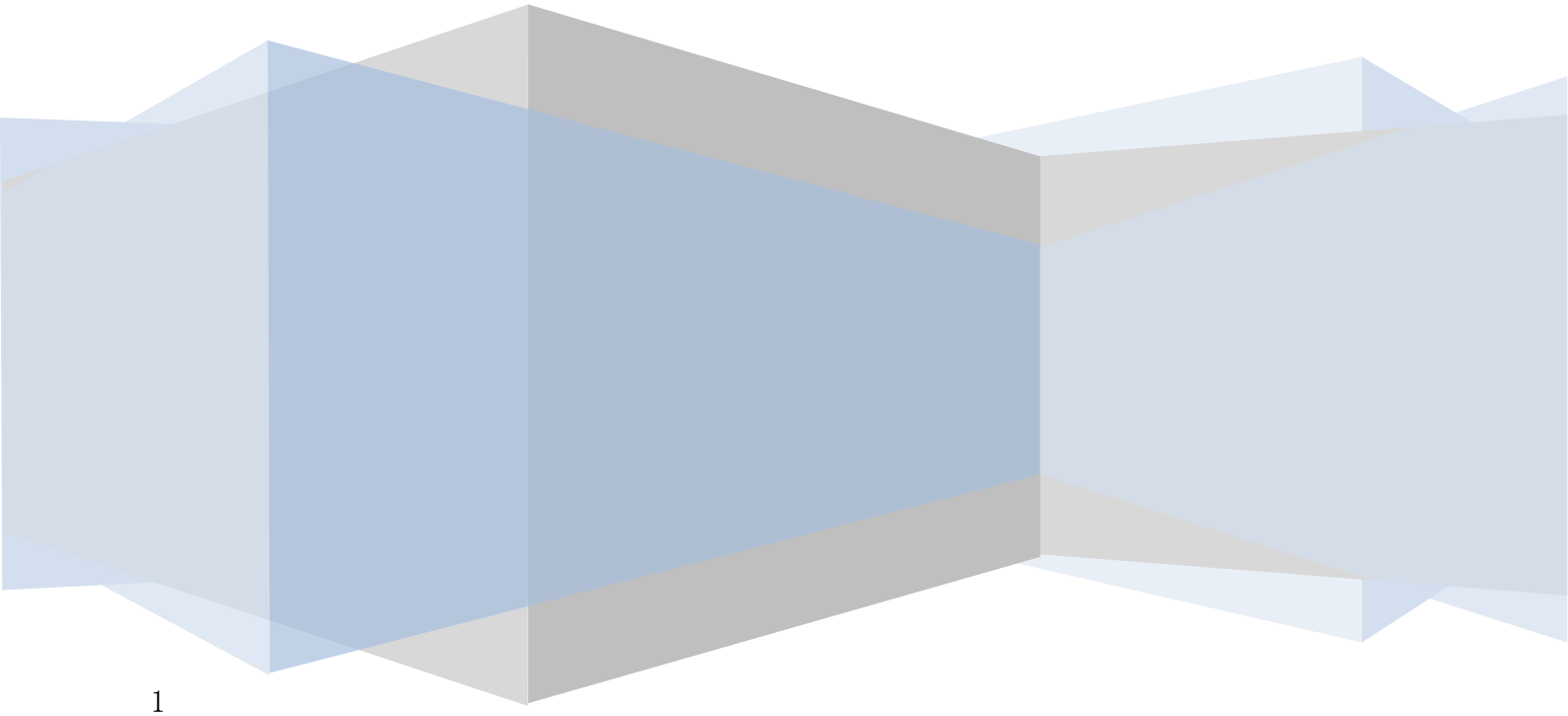
SEMESTER PROJECT REPORT ON DATABASE SYNCHRONIZATION

(REPLICATION and CLUSTERS IN MYSQL)

SUBMITTED TO:
SAURABH BARJATIYA

SUBMITTED BY:
DEEPAK SHRIVASTAVA (200905007)



ACKNOWLEDGEMENT

This Project has been made for the partial fulfillment of the MTech (C.S.E) III rd semester. My heartiest indebtedness to Saurabh Barjatiya Sir, project guide and lecturer who made available all the necessary faculties for the successful completion of the project and guided me through out the project.

I also wish to express my deep sense of gratitude to Prof. Shatrunjay Rawat Sir, and all my colleagues for their continuous and tireless support and advise, not only during the course of this project, but also during other times, whose active association, constant encouragement, untiring labor and generous efforts could enable me to lay out the work of this project.

DEEPAK SHRIVASTAVA 200905007

INDEX

1. OBJECTIVE: objective of this project is to synchronize the various database mirrors located in various locations and to try load balancing among these servers so that read queries can be handled by any of the server including mirrors and primary server and update/insert queries will only be handled by master server.

This report illustrates the steps to set up replication process between mysql database server (master) and mysql database servers (slaves). Which is used to synchronize the master and slave databases automatically.

2. REPLICATION: replication is the process of creating same replicas of the database as in master server, in all slave servers. Replication enables data from one MySQL database server (the master) to be replicated to one or more MySQL database servers (the slaves). Replication is asynchronous - slaves need not to connect permanently to receive updates from the master. This means that updates can occur over long-distance connections and even over temporary or intermittent connections such as a dial-up service. Depending on the configuration, we can replicate all databases, selected databases, or even selected tables within a database.

Replication between servers in MySQL is based on the binary logging mechanism. The MySQL instance operating as the master (the source of the database changes) writes updates and changes as “events” to the binary log. The information in the binary log is stored in different logging formats according to the database changes being recorded. Slaves are configured to read the binary log from the master and to execute the events in the binary log on the slave's local database.

The master is “dumb” in this scenario. Once binary logging has been enabled, all statements are recorded in the binary log. Each slave receives a copy of the entire contents of the binary log. It is the responsibility of the slave to decide which statements in the binary log should be executed; we cannot configure the master to log only certain events. If we do not specify otherwise, all events in the master binary log are executed on the slave. If required, we can configure the slave to process only events that apply to particular databases or tables.

Each slave keeps a record of the binary log coordinates: The file name and position within the file that it has read and processed from the master. This means that multiple slaves can be connected to the master and executing different parts of the same binary log. Because the slaves control this process, individual slaves can be connected and disconnected from the server without affecting the master's operation. Also, because each slave remembers the position within the binary log, it is possible for slaves to be disconnected, reconnect and then “catch up” by continuing from the recorded position.

3. STEPS TO SET UP REPLICATION:

i. Replication Master Configuration:

On a replication master, we must enable binary logging and establish a unique server ID. Binary logging must be enabled on the master because the binary log is the basis for sending data changes from the master to its slaves. If binary logging is not enabled, replication will not be possible.

Each server within a replication group must be configured with a unique server ID. This ID is used to identify individual servers within the group, and must be a positive integer between 1 and 231

To configure the binary log and server ID options, we will need to shut down your MySQL server and edit the my.cnf or config.ini file. Add the following options to the configuration file within the **[mysqld]** section. If these options already exist, but are commented out, uncomment the options and alter them according to our needs. For example, to enable binary logging using a log file name prefix of **mysql-bin**, and configure a server ID of 1, use these lines

[mysqld]

Log-bin=mysql-bin

Server-id=1

After making the changes, restart the server. Server-id should be unique to each server other wise replication will not work. Other than that we have to also ensure that skip networking option is disabled but now days instead of skip only default is to listen on local host. So we have to comment the line

bind-address =127.0.0.1

In the file my.cnf this will enable the master and slave to communicate.

ii. Replication Slave Configuration:

On a replication slave, we have to establish a unique server ID. If this has not already been done, this part of slave setup requires a server restart.

Again we have to edit the file my.cnf and against [mysqld] tag we have to write

[mysqld]

Server-id = 2

After making the changes we have to restart the server by ‘service mysql restart’ command. All those servers which are going to be working as slaves should have unique server id.

Each slave must connect to the master using a MySQL user name and password, so there must be a user account on the master that the slave can use to connect. Any account can be used for this operation, providing it has been granted the REPLICATION SLAVE privilege. We may wish to create a different account for each slave, or connect to the master using the same account for each slave.

we need not create an account specifically for replication. However, we should be aware that the user name and password will be stored in plain text within the master.info file. Therefore, we may want to create a separate account that has privileges only for the replication process, to minimize the possibility of compromise to other accounts.

To create a new account, use CREATE USER. To grant this account the privileges required for replication, use the GRANT statement. If we create an account solely for the purposes of replication, that account needs only the REPLICATION SLAVE privilege. For example, to set up a new user, repl that can connect for replication from any host within the mydomain.com domain issue these statements on the master:

mysql> CREATE USER 'repl'@'%.mydomain.com' IDENTIFIED BY 'slavepass';

mysql> GRANT REPLICATION SLAVE ON *.* TO 'repl'@'%.mydomain.com';

iii. Obtaining the Replication Master Binary Log Coordinates

To configure replication on the slave we must determine the master's current coordinates within its binary log. We will need this information so that when the slave starts the replication process, it is able to start processing events from the binary log at the correct point.

If we have existing data on our master that we want to synchronize on our slaves before starting the replication process, we must stop processing statements on the master, and then obtain its current binary log coordinates and dump its data, before permitting the master to continue executing statements. If we do not stop the execution of statements, the data dump and the master status information that we use will not match and we will end up with inconsistent or corrupted databases on the slaves.

To obtain the master binary log coordinates, follow these steps:

Start a session on the master by connecting to it with the command-line client, and flush all tables and block write statements by executing the FLUSH TABLES WITH READ LOCK statement:

mysql> FLUSH TABLES WITH READ LOCK;

In a different session on the master, use the SHOW MASTER STATUS statement to determine the current binary log file name and position:

```
mysql > SHOW MASTER STATUS;
+-----------------------+----------+--------------+------------------+-----------
| File                  | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----------------------+----------+--------------+------------------+------
| mysql-bin.000003      | 73       | test         |  manual,mysql    |
+-----------------------+----------+--------------+------------------+-----------
```

The File column shows the name of the log file and Position shows the position within the file. In this example, the binary log file is mysql-bin.000003 and the position is 73. Record these values. We need them later when we are setting up the slave. They represent the replication coordinates at which the slave should begin processing new updates from the master.

If the master has been running previously without binary logging enabled, the log file name and position values displayed by SHOW MASTER STATUS or mysqldump --master-data will be empty. In that case, the values that we need to use later when specifying the slave's log file and position are the empty string (") and 4.

We now have the information we need to enable the slave to start reading from the binary log in the correct place to start replication.

If we have existing data that needs be to synchronized with the slave before we start replication, leave the client running so that the lock remains in place and then create a data snap shot or raw data snap shot. The idea here is to prevent any further changes so that the data copied to the slaves is in synchrony with the master.

If we are setting up a brand new master and slave replication group, we can exit the first session to release the read lock.

iv. Creating a Data Snapshot Using mysqldump

One way to create a snapshot of the data in an existing master database is to use the **mysqldump** tool. Once the data dump has been completed, we then import this data into the slave before starting the replication process. If we are establishing new master then there is no need to perform this step.

To obtain a snapshot of the data using **mysqldump**:

If we have not already locked the tables on the server to prevent statements that update data from executing:

Start a session on the server by connecting to it with the command-line client, and flush all tables and block write statements by executing the FLUSH TABLES WITH READ LOCK statement:

mysql> **FLUSH TABLES WITH READ LOCK;**

In another session, use **mysqldump** to create a dump either of all the databases we want to replicate, or of selected individual databases. For example:

shell> **mysqldump --all-databases --lock-all-tables >dbdump.db**

for all the databases or

shell> mysqldump -u user -p db-name[table_name] > dbdump.db

for single database/table.

An alternative to using a bare dump, is to use the --master-data option, which automatically appends the CHANGE MASTER TO statement required on the slave to start the replication process.

shell> **mysqldump --all-databases --master-data >dbdump.db**

In the client where we acquired the read lock, release the lock:

mysql> **UNLOCK TABLES;**

Alternatively, exit the first session to release the read lock.

When choosing databases to include in the dump, remember that we will need to filter out databases on each slave that we do not want to include in the replication process.

We will need either to copy the dump file to the slave, or to use the file from the master when connecting remotely to the slave to import the data.

v. Dumping data in slave server:

once we have taken the snapshot of the database we have to dump that data to the slave server. To do that we have to copy dump file for example dbdump.db to slave by any means possible like ssh or scp. After copying dump file into slave we have to run

mysql < dbdump.db

After that we will start the server and set the master configuration in slave.

vi. Setting the Master Configuration on the Slave.

To set up the slave to communicate with the master for replication, we must tell the slave the necessary connection information. To do this, execute the following statement on the slave, replacing the option values with the actual values relevant to our system:

```
mysql> CHANGE MASTER TO
    -> MASTER_HOST='master_host_name',
    -> MASTER_USER='replication_user_name',
    -> MASTER_PASSWORD='replication_password',
    -> MASTER_LOG_FILE='recorded_log_file_name',
    -> MASTER_LOG_POS=recorded_log_position;
```

in place of recorded_log_position we have to write the position we obtained using show master status which represents the position from which slave has to read the master binary log and in place of recorded_log_file_name we will write master binary log file name.

Replication cannot use Unix socket files. We must be able to connect to the master MySQL server using TCP/IP.

The CHANGE MASTER TO statement has other options as well. For example, it is possible to set up secure replication using SSL.

vii. Introducing Additional Slaves to an Existing Replication Environment

To add another slave to an existing replication configuration, we can do so without stopping the master. Instead, set up the new slave by making a copy of an existing slave, except that we configure the new slave with a different server-id value. Or we can also use the same process as we did to make the first slave.

To duplicate an existing slave:

Shut down the existing slave:

```
shell> mysqladmin shutdown
```

Copy the data directory from the existing slave to the new slave. We can do this by creating an archive using tar or WinZip, or by performing a direct copy using a tool such as cp or rsync. Ensure that we also copy the log files and relay log files.

A common problem that is encountered when adding new replication slaves is that the new slave

fails with a series of warning and error messages like these:

071118 16:44:10 [Warning] Neither --relay-log nor --relay-log-index were used; so

replication may break when this MySQL server acts as a slave and has his hostname

changed!! Please use '--relay-log=new_slave_hostname-relay-bin' to avoid this problem.

071118 16:44:10 [ERROR] Failed to open the relay log './old_slave_hostname-relay-bin.003525'

(relay_log_pos 22940879)

071118 16:44:10 [ERROR] Could not find target log during relay log initialization

071118 16:44:10 [ERROR] Failed to initialize the master info structure

This is due to the fact that, if the --relay-log option is not specified, the relay log files contain the host name as part of their file names.

To avoid this problem, use the same value for --relay-log on the new slave that was used on the existing slave. (If this option was not set explicitly on the existing slave, use existing_slave_hostname-relay-bin.) If this is not feasible, copy the existing slave's relay log index file to the new slave and set the --relay-log-index option on the new slave to match what was used on the existing slave. (If this option was not set explicitly on the existing slave, use existing_slave_hostname-relay-bin.index.) Alternatively—if we have already tried to start the new slave (after following the remaining steps in this section) and have encountered errors like those described previously—then perform the following steps:

If we have not already done so, issue a STOP SLAVE on the new slave.

If we have already started the existing slave again, issue a STOP SLAVE on the existing slave as well.

Copy the contents of the existing slave's relay log index file into the new slave's relay log index file, making sure to overwrite any content already in the file.

Proceed with the remaining steps in this section.

Copy the master.info and relay-log.info files from the existing slave to the new slave if they were not located in the data directory. These files hold the current log coordinates for the master's binary log and the slave's relay log.

Start the existing slave.

On the new slave, edit the configuration and give the new slave a unique server-id not used by the master or any of the existing slaves.

Start the new slave. The slave will use the information in its master.info file to start the replication

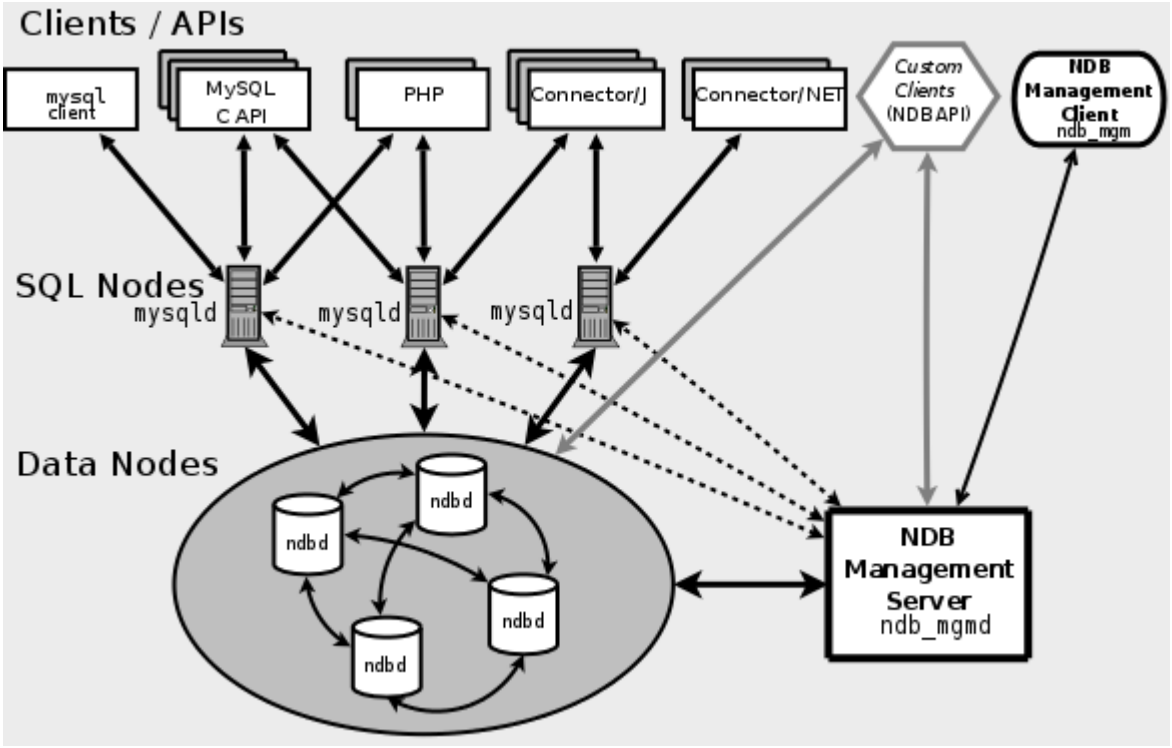
process.

4. Problems With Replication:

The major problem with replication is that it is asynchronous which means all the slave servers are not necessarily gets updated simultaneously. So at a given time all the servers might not have same content which can lead to give inconsistent results and the scenarios in which replication can be used are very limited such as Facebook etc. where we can afford delayed updation of all the servers. The solution of this problem could be Mysql cluster which is a synchronous version of mysql in which all the data nodes get updated simultaneously by means of shared memory details of mysql cluster are as follows.

5. Mysql CLUSTER Overview:

MySQL Cluster is a technology that enables clustering of in-memory databases in a shared-nothing system. The shared-nothing architecture enables the system to work with very inexpensive hardware, and with a minimum of specific requirements for hardware or software.
MySQL Cluster is designed not to have any single point of failure. In a shared-nothing system, each component is expected to have its own memory and disk, and the use of shared storage mechanisms such as network shares, network file systems, and SANs is not recommended or supported.
MySQL Cluster integrates the standard MySQL server with an in-memory clustered storage engine called NDB (which stands for “Network Database”). In our documentation, the term NDB refers to the part of the setup that is specific to the storage engine, whereas “MySQL Cluster” refers to the combination of one or more MySQL servers with the NDB storage engine.
A MySQL Cluster consists of a set of computers, known as hosts, each running one or more processes. These processes, known as nodes, may include MySQL servers (for access to NDB data), data nodes (for storage of the data), one or more management servers, and possibly other specialized data access programs. The relationship of these components in a MySQL Cluster is shown here:



All these programs work together to form a MySQL Cluster When data is stored by the NDB storage engine, the tables (and table data) are stored in the data nodes. Such tables are directly accessible from all other MySQL servers (SQL nodes) in the cluster. Thus, in a payroll application storing data in a cluster, if one application updates the salary of an employee, all other MySQL servers that query this data can see this change immediately.

Although a MySQL Cluster SQL node uses the **mysqld** server daemon, it differs in a number of critical respects from the **mysqld** binary supplied with the MySQL 5.1 distributions, and the two versions of **mysqld** are not interchangeable.

In addition, a MySQL server that is not connected to a MySQL Cluster cannot use the NDB storage engine and cannot access any MySQL Cluster data.

The data stored in the data nodes for MySQL Cluster can be mirrored; the cluster can handle failures of individual data nodes with no other impact than that a small number of transactions are aborted due to losing the transaction state. Because transactional applications are expected to handle transaction failure, this should not be a source of problems.

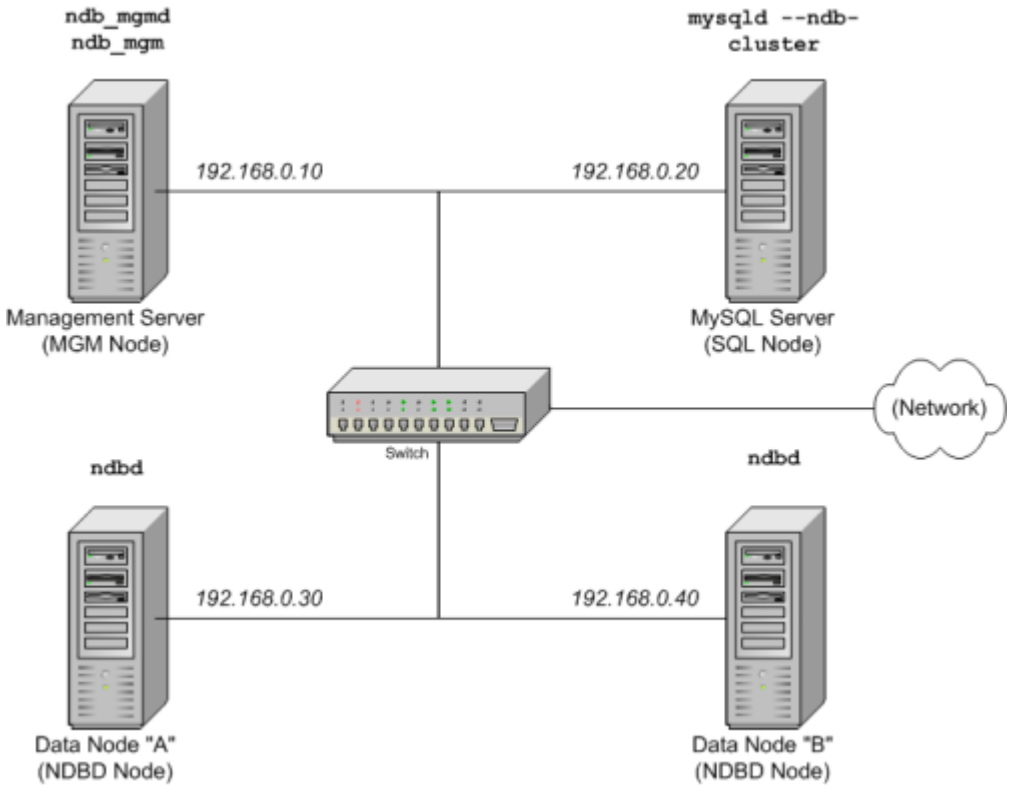
Individual nodes can be stopped and restarted, and can then rejoin the system (cluster). Rolling restarts (in which all nodes are restarted in turn) are used in making configuration changes and software upgrades.

6. How to Setup A CLUSTER:

For example we will explain the procedures to set up a cluster consisting of 4 nodes, each on separate hosts, and each with a fixed network address on a typical Ethernet network as shown here:

Node	IP Address
Management (MGMD) node	192.168.0.10
MySQL server (SQL) node	192.168.0.20
Data (NDBD) node "A"	192.168.0.30
Data (NDBD) node "B"	192.168.0.40

This may be made clearer in the following diagram:



In the interest of simplicity (and reliability), this report uses only numeric IP addresses. However, if DNS resolution is available on your network, it is possible to use host names in lieu of IP addresses in configuring Cluster.

Installation of different nodes of cluster:

Each MySQL Cluster host computer running an SQL node must have installed on it a MySQL binary. For management nodes and data nodes, it is not necessary to install the MySQL server binary, but management nodes require the management server daemon (**ndb_mgmd**) and data nodes require the data node daemon (**ndbd**; in MySQL Cluster NDB 7.0 and later, you can use **ndbmtd** instead). It is also a good idea to install the management client (**ndb_mgm**) on the

management server host. This section covers the steps necessary to install the correct binaries for each type of Cluster node.

You have to download the latest version of mysql cluster from the mysql site's download section for 32/64 bit versions in tar.gz form.

Data and SQL Node Installation: .tar.gz Binary. On each of the machines designated to host data or SQL nodes, perform the following steps as the system root user:

Check your /etc/passwd and /etc/group files (or use whatever tools are provided by your operating system for managing users and groups) to see whether there is already a mysql group and mysql user on the system. Some OS distributions create these as part of the operating system installation process. If they are not already present, create a new mysql user group, and then add a mysql user to this group:

shell> **groupadd mysql**

shell> **useradd -g mysql mysql**

The syntax for **useradd** and **groupadd** may differ slightly on different versions of Unix, or they may have different names such as **adduser** and **addgroup**.

Change location to the directory containing the downloaded file, unpack the archive, and create a symlink to the mysql directory named mysql. Note that the actual file and directory names vary according to the MySQL Cluster version number.

shell> **cd /var/tmp**

shell> **tar -C /usr/local -xzvf mysql-cluster-gpl-7.1.10-linux-i686-glibc23.tar.gz**

shell> **ln -s /usr/local/mysql-cluster-gpl-7.1.10-linux-i686-glibc23.tar.gz /usr/local/mysql**

Change location to the mysql directory and run the supplied script for creating the system databases:

shell> **cd mysql**

shell> **scripts/mysql_install_db --user=mysql**

Set the necessary permissions for the MySQL server and data directories:

shell> **chown -R root .**

shell> **chown -R mysql data**

shell> **chgrp -R mysql .**

Note that the data directory on each machine hosting a data node is /usr/local/mysql/data. This

piece of information is essential when configuring the management node. Copy the MySQL startup script to the appropriate directory, make it executable, and set it to start when the operating system is booted up:

shell> **cp support-files/mysql.server /etc/rc.d/init.d/**

shell> **chmod +x /etc/rc.d/init.d/mysql.server**

shell> **chkconfig --add mysql.server**

(The startup scripts directory may vary depending on your operating system and version—for example, in some Linux distributions, it is /etc/init.d.)

Here we use Red Hat's **chkconfig** for creating links to the startup scripts; use whatever means is appropriate for this purpose on your operating system and distribution, such as **update-rc.d** on Debian.

Remember that the preceding steps must be repeated on each machine where an SQL node is to reside.

Data node installation: RPM Files. On a computer that is to host a cluster data node it is necessary to install only the **NDB Cluster - Storage engine** RPM. To do so, copy this RPM to the data node host, and run the following command as the system root user, replacing the name shown for the RPM as necessary to match that of the RPM downloaded from the MySQL web site:

shell> **rpm -Uhv MySQL-Cluster-gpl-storage-7.1.10-0.sles10.i586.rpm**

The previous command installs the MySQL Cluster data node binary (**ndbd**) in the /usr/sbin directory.

Management node installation: .tar.gz binary. Installation of the management node does not require the **mysqld** binary. Only the MySQL Cluster management server (**ndb_mgmd**) is required; you most likely want to install the management client (**ndb_mgm**) as well. Both of these binaries also be found in the .tar.gz archive. Again, we assume that you have placed this archive in /var/tmp.

As system root (that is, after using **sudo**, **su root**, or your system's equivalent for temporarily assuming the system administrator account's privileges), perform the following steps to install **ndb_mgmd** and **ndb_mgm** on the Cluster management node host:

1. Change location to the /var/tmp directory, and extract the **ndb_mgm** and **ndb_mgmd** from the archive into a suitable directory such as /usr/local/bin:
2. shell> **cd /var/tmp**
3. shell> **tar -zxvf mysql-5.1.51-ndb-7.1.10-linux-i686-glibc23.tar.gz**
4. shell> **cd mysql-5.1.51-ndb-7.1.10-linux-i686-glibc23**
5. shell> **cp bin/ndb_mgm* /usr/local/bin**

(You can safely delete the directory created by unpacking the downloaded archive, and the files it contains, from /var/tmp once **ndb_mgm** and **ndb_mgmd** have been copied to the executables directory.)

6. Change location to the directory into which you copied the files, and then make both of them executable:
7. shell> **cd /usr/local/bin**
8. shell> **chmod +x ndb_mgm***

7. MySQL Cluster Multi-Computer Configuration:

For our four-node, four-host MySQL Cluster, it is necessary to write four configuration files, one per node host.

- Each data node or SQL node requires a my.cnf file that provides two pieces of information: a connectstring that tells the node where to find the management node, and a line telling the MySQL server on this host (the machine hosting the data node) to enable the NDBCLUSTER storage engine.

For more information on connectstrings,.

- The management node needs a config.ini file telling it how many replicas to maintain, how much memory to allocate for data and indexes on each data node, where to find the data nodes, where to save data to disk on each data node, and where to find any SQL nodes.

Configuring the storage and SQL nodes. The my.cnf file needed for the data nodes is fairly simple. The configuration file should be located in the /etc directory and can be edited using any text editor. (Create the file if it does not exist.) For example:

shell> **vi /etc/my.cnf**

Note

We show **vi** being used here to create the file, but any text editor should work just as well.

For each data node and SQL node in our example setup, my.cnf should look like this:

[mysqld]

Options for mysqld process:

ndbcluster # run NDB storage engine

ndb-connectstring=192.168.0.10 # location of management server

[mysql_cluster]

Options for ndbd process:

ndb-connectstring=192.168.0.10 # location of management server

After entering the preceding information, save this file and exit the text editor. Do this for the machines hosting data node “A”, data node “B”, and the SQL node.

Important

Once you have started a **mysqld** process with the NDBCLUSTER and ndb-connectstring parameters in the [mysqld] in the my.cnf file as shown previously, you cannot execute any CREATE TABLE or ALTER TABLE statements without having actually started the cluster. Otherwise, these statements will fail with an error. This is by design.

Configuring the management node. The first step in configuring the management node is to create the directory in which the configuration file can be found and then to create the file itself. For example (running as root):

shell> **mkdir /var/lib/mysql-cluster**

shell> **cd /var/lib/mysql-cluster**

shell> **vi config.ini**

For our representative setup, the config.ini file should read as follows:

[ndbd default]

Options affecting ndbd processes on all data nodes:

NoOfReplicas=2 # Number of replicas

DataMemory=80M # How much memory to allocate for data storage

IndexMemory=18M # How much memory to allocate for index storage

For DataMemory and IndexMemory, we have used the

default values. Since the “world” database takes up

only about 500KB, this should be more than enough for

```
            # this example Cluster setup.
[tcp default]
# TCP/IP options:
portnumber=2202   # This the default; however, you can use any
            # port that is free for all the hosts in the cluster
            # Note: It is recommended that you do not specify the port
            # number at all and simply allow the default value to be used
            # instead
[ndb_mgmd]
# Management process options:
hostname=192.168.0.10          # Hostname or IP address of MGM node
datadir=/var/lib/mysql-cluster  # Directory for MGM node log files
[ndbd]
# Options for data node “A”:
                        # (one [ndbd] section per data node)
hostname=192.168.0.30          # Hostname or IP address
datadir=/usr/local/mysql/data   # Directory for this data node’s data files
[ndbd]
# Options for data node “B”:
hostname=192.168.0.40          # Hostname or IP address
datadir=/usr/local/mysql/data   # Directory for this data node’s data files
[mysqld]
# SQL node options:
hostname=192.168.0.20          # Hostname or IP address
                        # (additional mysqld connections can be
                        # specified for this node for various
                        # purposes such as running ndb_restore)
```

8. Initial Startup of MySQL Cluster

Starting the cluster is not very difficult after it has been configured. Each cluster node process must be started separately, and on the host where it resides. The management node should be started first, followed by the data nodes, and then finally by any SQL nodes:

On the management host, issue the following command from the system shell to start the management node process:

shell> **ndb_mgmd –f /var/lib/mysql-cluster/config.ini**

Note

ndb_mgmd must be told where to find its configuration file, using the –f or –config-file option.

On each of the data node hosts, run this command to start the **ndbd** process:

shell> **ndbd**

If you used RPM files to install MySQL on the cluster host where the SQL node is to reside, you can (and should) use the supplied startup script to start the MySQL server process on the SQL node.

If all has gone well, and the cluster has been set up correctly, the cluster should now be operational. You can test this by invoking the **ndb_mgm** management node client. The output should look like that shown here, although you might see some slight differences in the output depending upon the exact version of MySQL that you are using:

shell> **ndb_mgm**

-- NDB Cluster – Management Client –

ndb_mgm> **SHOW**

Connected to Management Server at: localhost:1186

Cluster Configuration

[ndbd(NDB)] 2 node(s)

id=2 @192.168.0.30 (Version: 5.1.51-ndb-6.3.40, Nodegroup: 0, Master)

id=3 @192.168.0.40 (Version: 5.1.51-ndb-6.3.40, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)

id=1 @192.168.0.10 (Version: 5.1.51-ndb-6.3.40)

[mysqld(API)] 1 node(s)

id=4 @192.168.0.20 (Version: 5.1.51-ndb-6.3.40)

The SQL node is referenced here as [mysqld(API)], which reflects the fact that the **mysqld** process is acting as a MySQL Cluster API node.

Note

The IP address shown for a given MySQL Cluster SQL or other API node in the output of SHOW is the address used by the SQL or API node to connect to the cluster data nodes, and not to any management node.

You should now be ready to work with databases, tables, and data in MySQL Cluster.

9. Loading Sample Data into MySQL Cluster and Performing Queries:

Working with data in MySQL Cluster is not much different from doing so in MySQL without clustering support. There are two points to keep in mind:

For a table to be replicated in the cluster, it must use the NDBCLUSTER storage engine. To specify this, use the ENGINE=NDBCLUSTER or ENGINE=NDB option when creating the table:

CREATE TABLE tbl_name (col_name column_definitions) ENGINE=NDBCLUSTER;

Alternatively, for an existing table that uses a different storage engine, use ALTER TABLE to change the table to use NDBCLUSTER:

ALTER TABLE tbl_name ENGINE=NDBCLUSTER;

Each NDBCLUSTER table must have a primary key. If no primary key is defined by the user when a table is created, the NDBCLUSTER storage engine automatically generates a hidden one.

Note

This hidden key takes up space just as does any other table index. It is not uncommon to encounter problems due to insufficient memory for accommodating these automatically created indexes.)

If you are importing tables from an existing database using the output of **mysqldump**, you can open the SQL script in a text editor and add the ENGINE option to any table creation statements, or replace any existing ENGINE options. Suppose that you have the world sample database on another MySQL server that does not support MySQL Cluster, and you want to export the City table:

shell> **mysqldump --add-drop-table world City > city_table.sql**

The resulting city_table.sql file will contain this table creation statement (and the INSERT

statements necessary to import the table data):

```
DROP TABLE IF EXISTS `City`;
CREATE TABLE `City` (
  `ID` int(11) NOT NULL auto_increment,
  `Name` char(35) NOT NULL default '',
  `CountryCode` char(3) NOT NULL default '',
  `District` char(20) NOT NULL default '',
  `Population` int(11) NOT NULL default '0',
  PRIMARY KEY  (`ID`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
INSERT INTO `City` VALUES (1,'Kabul','AFG','Kabol',1780000);
INSERT INTO `City` VALUES (2,'Qandahar','AFG','Qandahar',237500);
INSERT INTO `City` VALUES (3,'Herat','AFG','Herat',186800);
(remaining INSERT statements omitted)
```

You need to make sure that MySQL uses the NDBCLUSTER storage engine for this table. There are two ways that this can be accomplished. One of these is to modify the table definition before importing it into the Cluster database. Using the City table as an example, modify the ENGINE option of the definition as follows:

```
DROP TABLE IF EXISTS `City`;
CREATE TABLE `City` (
  `ID` int(11) NOT NULL auto_increment,
  `Name` char(35) NOT NULL default '',
  `CountryCode` char(3) NOT NULL default '',
  `District` char(20) NOT NULL default '',
  `Population` int(11) NOT NULL default '0',
  PRIMARY KEY  (`ID`)
) ENGINE=NDBCLUSTER DEFAULT CHARSET=latin1;
INSERT INTO `City` VALUES (1,'Kabul','AFG','Kabol',1780000);
```

INSERT INTO `City` VALUES (2,'Qandahar','AFG','Qandahar',237500);

INSERT INTO `City` VALUES (3,'Herat','AFG','Herat',186800);

(remaining INSERT statements omitted)

This must be done for the definition of each table that is to be part of the clustered database. The easiest way to accomplish this is to do a search-and-replace on the file that contains the definitions and replace all instances of TYPE=engine_name or ENGINE=engine_name with ENGINE=NDBCLUSTER. If you do not want to modify the file, you can use the unmodified file to create the tables, and then use ALTER TABLE to change their storage engine. The particulars are given later in this section.

Assuming that you have already created a database named world on the SQL node of the cluster, you can then use the **mysql** command-line client to read city_table.sql, and create and populate the corresponding table in the usual manner:

shell> **mysql world < city_table.sql**

It is very important to keep in mind that the preceding command must be executed on the host where the SQL node is running (in this case, on the machine with the IP address 192.168.0.20).

To create a copy of the entire world database on the SQL node, use **mysqldump** on the noncluster server to export the database to a file named world.sql; for example, in the /tmp directory. Then modify the table definitions as just described and import the file into the SQL node of the cluster like this:

shell> **mysql world < /tmp/world.sql**

If you save the file to a different location, adjust the preceding instructions accordingly.

Running SELECT queries on the SQL node is no different from running them on any other instance of a MySQL server. To run queries from the command line, you first need to log in to the MySQL Monitor in the usual way (specify the root password at the Enter password: prompt):

shell> **mysql -u root -p**

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 1 to server version: 5.1.51-ndb-6.2.19

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>

We simply use the MySQL server's root account and assume that you have followed the standard

security precautions for installing a MySQL server, including setting a strong root password.

It is worth taking into account that Cluster nodes do not make use of the MySQL privilege system when accessing one another. Setting or changing MySQL user accounts (including the root account) effects only applications that access the SQL node, not interaction between nodes. If you did not modify the ENGINE clauses in the table definitions prior to importing the SQL script, you should run the following statements at this point:

mysql> **USE world;**

mysql> **ALTER TABLE City ENGINE=NDBCLUSTER;**

mysql> **ALTER TABLE Country ENGINE=NDBCLUSTER;**

mysql> **ALTER TABLE CountryLanguage ENGINE=NDBCLUSTER;**

Selecting a database and running a **SELECT** query against a table in that database is also accomplished in the usual manner, as is exiting the MySQL Monitor:

mysql> **USE world;**

mysql> **SELECT Name, Population FROM City ORDER BY Population DESC LIMIT 5;**

+----------+------------+

| Name | Population |

+----------+------------+

| Bombay | 10500000 |

| Seoul | 9981619 |

| São Paulo | 9968485 |

| Shanghai | 9696300 |

| Jakarta | 9604900 |

+----------+------------+

5 rows in set (0.34 sec)

mysql> **\q**

Bye

shell>

Applications that use MySQL can employ standard APIs to access NDB tables. It is important to remember that your application must access the SQL node, and not the management or data

nodes. This brief example shows how we might execute the SELECT statement just shown by using the PHP 5.X mysqli extension running on a Web server elsewhere on the network:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type"
        content="text/html; charset=iso-8859-1">
  <title>SIMPLE mysqli SELECT</title>
</head>
<body>
<?php
  # connect to SQL node:
  $link = new mysqli('192.168.0.20', 'root', 'root_password', 'world');
  # parameters for mysqli constructor are:
  #   host, user, password, database
  if( mysqli_connect_errno() )
    die("Connect failed: " . mysqli_connect_error());
  $query = "SELECT Name, Population
          FROM City
          ORDER BY Population DESC
          LIMIT 5";
  # if no errors...
  if( $result = $link->query($query) )
  {
?>
<table border="1" width="40%" cellpadding="4" cellspacing ="1">
```

```
  <tbody>
  <tr>
    <th width="10%">City</th>
    <th>Population</th>
  </tr>
<?
    # then display the results...
    while($row = $result->fetch_object())
      printf("<tr>\n  <td align=\"center\">%s</td><td>%d</td>\n</tr>\n",
            $row->Name, $row->Population);
?>
  </tbody
</table>
<?
  # ...and verify the number of rows that were retrieved
    printf("<p>Affected rows: %d</p>\n", $link->affected_rows);
  }
  else
    # otherwise, tell us what went wrong
    echo mysqli_error();
  # free the result set and the mysqli connection object
  $result->close();
  $link->close();
?>
</body>
</html>
```

We assume that the process running on the Web server can reach the IP address of the SQL

node.

In a similar fashion, you can use the MySQL C API, Perl-DBI, Python-mysql, or MySQL Connectors to perform the tasks of data definition and manipulation just as you would normally with MySQL.

10. Safe Shutdown and Restart of MySQL Cluster:

To shut down the cluster, enter the following command in a shell on the machine hosting the management node:

shell> **ndb_mgm -e shutdown**

The -e option here is used to pass a command to the **ndb_mgm** client from the shell. The command causes the **ndb_mgm**, **ndb_mgmd**, and any **ndbd** or **ndbmtd** processes to terminate gracefully. Any SQL nodes can be terminated using **mysqladmin shutdown** and other means. On Windows platforms, assuming that you have installed the SQL node as a Windows service, you can use **NET STOP MYSQL**.

To restart the cluster on Unix platforms, run these commands:

On the management host (192.168.0.10 in our example setup):

shell> **ndb_mgmd -f /var/lib/mysql-cluster/config.ini**

On each of the data node hosts (192.168.0.30 and 192.168.0.40):

shell> **ndbd**

Use the **ndb_mgm** client to verify that both data nodes have started successfully.

On the SQL host (192.168.0.20):

shell> **mysqld_safe &**

On Windows platforms, assuming that you have installed all MySQL Cluster processes as Windows services using the default service names, you can restart the cluster as follows:

On the management host (192.168.0.10 in our example setup), execute the following command:

C:\> **NET START ndb_mgmd**

On each of the data node hosts (192.168.0.30 and 192.168.0.40), execute the following command:

C:\> **NET START ndbd**

On the management node host, use the **ndb_mgm** client to verify that the management node and both data nodes have started successfully On the SQL node host (192.168.0.20), execute the following command:

C:\> **NET START mysql**

In a production setting, it is usually not desirable to shut down the cluster completely. In many cases, even when making configuration changes, or performing upgrades to the cluster hardware or software (or both), which require shutting down individual host machines, it is possible to do so without shutting down the cluster as a whole by performing a rolling restart of the cluster. For more information about doing this.

11. Conclusion: As part of this project we learned various ways of database synchronization such as setting up replication among mysql master and slave servers which is asynchronous, easy, and fast we analyzed what are the problems of replication and situations where updates are to be simultaneously updated replication is not very useful. To counter these limitations we have mysql cluster version which is a synchronous version and is based on the basis of shared memory. It is difficult to configure and install but removes the limitations of replications. During the configuration of mysql cluster I faced few problems while running sql node sometimes locks are not freed on the socket so we have to take care of freeing locks before starting sql node. I tested the cluster with various conditions such as disconnecting sql nodes and data nodes abruptly while running and founfd the cluster still performs well and maintains the integrity and as soon as the sql node which was disconnected it updates itself with other sql nodes immediately. While configuring cluster we have to ensure that cluster is working properly via thorough testing. So this report provides hands on guide to establish and achieve replication and clusterization of servers.

12. REFERNCES:

1. http://dev.mysql.com/doc/refman/5.0/en/replication.html
2. http://dev.mysql.com/doc/refman/5.0/en/replication-solutions-scaleout.html
3. http://dev.mysql.com/doc/refman/5.0/en/replication-howto.html
4. http://www.cyberciti.biz/tips/howto-copy-mysql-database-remote-server.html
5. http://dev.mysql.com/doc/mysql-cluster-excerpt/5.1/en/index.html
6. http://blog.carlosgomez.net/2010/02/mysql-cluster-7-set-up-with-ubuntu.html